On the Application of the D* Search Algorithm to Time-Based Planning on Lattice Graphs

Martin Rufli Roland Siegwart
Autonomous Systems Lab, ETH Zurich, Tannenstrasse 3, CH-8092 Zurich, Switzerland
martin.rufli@mavt.ethz.ch rsiegwart@ethz.ch

***Abstract*— In this paper we present a multi-resolution state lattice, which operates in four dimensions, namely 2D position, heading, and velocity. The generation of such a lattice is described, resulting in an efficient (in terms of branching factor) and feasible (i.e. directly executable) set of edges, which can be searched on using any standard graph based planner.**

Furthermore, we introduce a novel heuristic, the *time-viable heuristic* with horizon T_n, which exploits the limited (but nonetheless extremely large) number of feasible motion combinations in a state lattice of bounded time and stores them in a look-up table. This heuristic then enables recently developed incremental planning algorithms, which typically start node expansion at the goal state (such as the various D* variants [1]) to be employed in time-based search, where the time of arrival is generally unknown a priori. We show, that by employing this technique, on average a comparable number of expanded states are to be expected for a given *initial planning* problem as when using forward searching algorithms (such as A* and variants [2, 3]), thus speeding up *re-planning* by up to two orders of magnitude as reported in [4].

***Index Terms*— Non-holonomic, time-based, motion planning, state lattice, time-viable heuristic**

I. Introduction

State lattices (applied to motion planning) have recently seen much attention in scenarios, where a preferable motion cannot be easily inferred from the environment (such as in planetary exploration [5, 6], off-road navigation, or in large-scale parking lots [7, 8, 9]). They are typically constructed by pruning an extremely large set of trajectory segments (generated with a suitably complex vehicle model via input or state-space sampling), down to a manageable subset. The underlying assumption is that through a sensible pruning policy, every single original trajectory segment is reconstructable by combining motion primitives from the subset alone, yielding a maximal deviation along the trajectory no larger than e_{max}. This error bound, and thus the state-space discretization, is justified by generally non-negligible sensor and actuator uncertainties present on robotic platforms. The resulting lattice is dependent on all assumptions made during vehicle modeling. Typically it is thus not portable between different vehicles, as it is based on the robot's steering configuration, actuator capabilities, and, if modeled, tire-road interaction. Poor portability and a long offline design phase are rewarded on-line with inherent executability of each lattice segment (see Fig. 1), and competitive search speed (for the case of a 3D lattice vs. a 2D grid based planner, see [10]). In the light of the above exposition it thus seems unclear why not more autonomous robots are equipped with lattice based planners.

Most literature on planning algorithms incorporating a state lattice is restricted to static, or quasi-static environments, where the lattice typically operates in a three-dimensional state-space (2D position and heading). Dynamic obstacles are typically treated in the same way as static obstacles, except for an inflated outline. Dynamic obstacle avoidance is then guaranteed via frequent re-planning (see i.e. [9]). These approaches work well in relatively slow, or open environments, but face issues in scenarios where some participants have a much higher maximal velocity, or where dynamic obstacles may block entire routes to the goal.

In general, more information than just the current pose of a dynamic obstacle can be extracted, however. Depending on the environment, predictions of variable quality can be deduced from an object's present and past motion. Approaches, which model the physics of the obstacle (see i.e. [12, 13]) typically achieve very precise short term predictions. On the other hand, approaches which model the intention of an agent fare better in the long term [14, 15, 16]. This additional information can be implemented into the planning stage, but requires the augmentation of the lattice with a velocity dimension (as was i.e. demonstrated in outdoor scenarios [9] and in indoor environments [11] to produce time-optimal as opposed to path length optimal solutions) and the addition of time to the state space (see [17]).

Full-range planning in nD (2D position, heading, steering angle, velocities, time, ...) is computationally intensive, however, and neither justified due to the uncertainty in motion prediction growing unbounded over time. Kushleyev et al. [17] recently presented a solution to this problem by introducing the time bounded lattice, a multi-resolution lattice, where all but two dimensions (2D position) are dropped as soon as motion prediction fidelity decreases below a certain threshold. In this paper we extend on this idea and enable the application of incremental re-planning algorithms to time-bounded lattices.

The remainder of this paper is organized as follows: section II reviews heuristic search applied to time-based planning on lattice graphs with a special emphasis on the introduction of our new time-viable heuristic. In Section III we demonstrate the effectiveness of our approach both by theoretical considerations and in simulated increasingly populated (and thus complex) scenarios. Finally, in Section IV we draw conclusions and sketch future work.

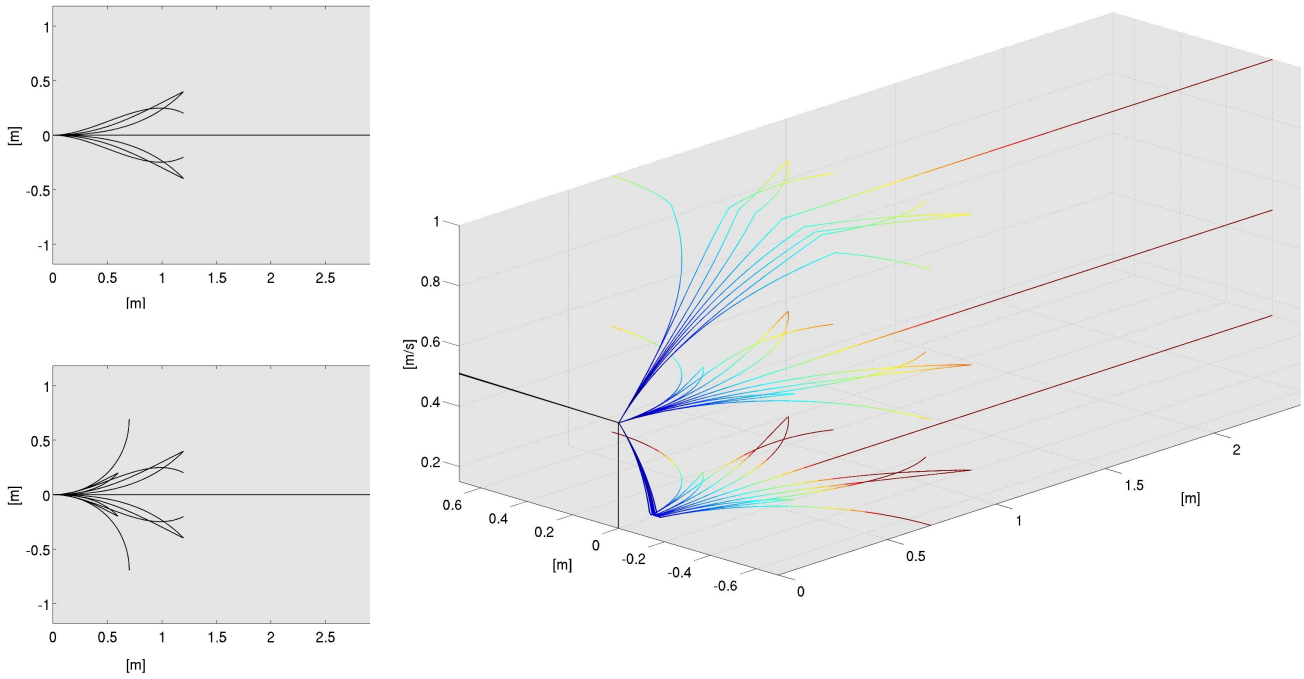

Fig. 1. 16-directional multi-resolution lattice (one of the 16 initial headings shown), as developed in [11]. Top Left: lattice low-resolution part. Bottom Left: high-resolution part. Some of the short straight segments are occluded. Right: full 4D (x, y, θ, v) view for one initial heading starting at a velocity of 0.5 m/s. Note that deceleration is higher than acceleration capability.

II. Applying D* in Combination with Motion Predictions

The efficiency by which D* repairs previously generated solutions (in many cases one to two orders of magnitude faster than re-planning from scratch [4]) renders it highly desirable for many deterministic graph search applications; especially so, as ever increasing computing power allows for higher-dimensional (and thus more complex) searches which better cope with various system and environmental constraints. By adding time to the state-space, the resulting motion is time-parametrized, and thus renders it impossible to initialize the goal state with the correct time of arrival, as this property is only known *after* computing the solution. A naive implementation would therefore estimate a time of arrival, plan, update the estimate, and continue until the estimate is confirmed. Such an implementation looses the speed boost over A* however. In this section, we present a solution to this problem in the form of a novel heuristic, the *time-viable heuristic*. We also describe the various components of our planner, with a special emphasis on adaptations due to the addition of motion predictions.

A. Graph

In [11], we developed a 4D multi-resolution lattice, where the resolution is adapted based on environmental (i.e. narrowness) and task characteristics (i.e. distance from the robot). It is generated by employing Catmul-Rom parametrized cubic splines and therefore allows for the control of end position, start and end heading, and, to some extent, maximal curvature along each segment. In the future we would like to incorporate quintic splines in order to specify curvature at edge boundaries and add rotational velocity to the state space. The lattice operates on a 0.1 m (high-res.) to 0.2 m (low-res.) grid, at 16 directions and with 6 velocities (0.0, 0.15, 0.5, 1.0 m/s, two rotational velocities). It has an average outdegree of close to 40. All successor edges of a single heading-velocity configuration are depicted in Fig. 1.

B. Heuristics

Heuristic based planner rely on a heuristic to guide them towards the goal state of the search. Well informed heuristics reduce planning time, as fewer states need to be expanded. The development of accurate heuristics is generally harder for higher-dimensional search spaces. We employ three complimentary heuristics, the combination of which is provably also a heuristic and generally very well informed.

1) 2D Heuristic: For a robot with nonholonomic constraints, a simple 2D Dijkstra search [18] on the lattice's underlying 2D grid yields accurate heuristic values far away from the robot position and in cases where environmental constraints severely limit robot mobility. As it relies on the environment, it cannot be precomputed offline. Depending on the environment size, such a computation can take several seconds, but only needs to be performed once per fixed goal state. For D* searches, this goal state (namely the current robot pose) changes continuously as the vehicle moves, however. We thus limit computation to a tunnel around a 2D A* search.

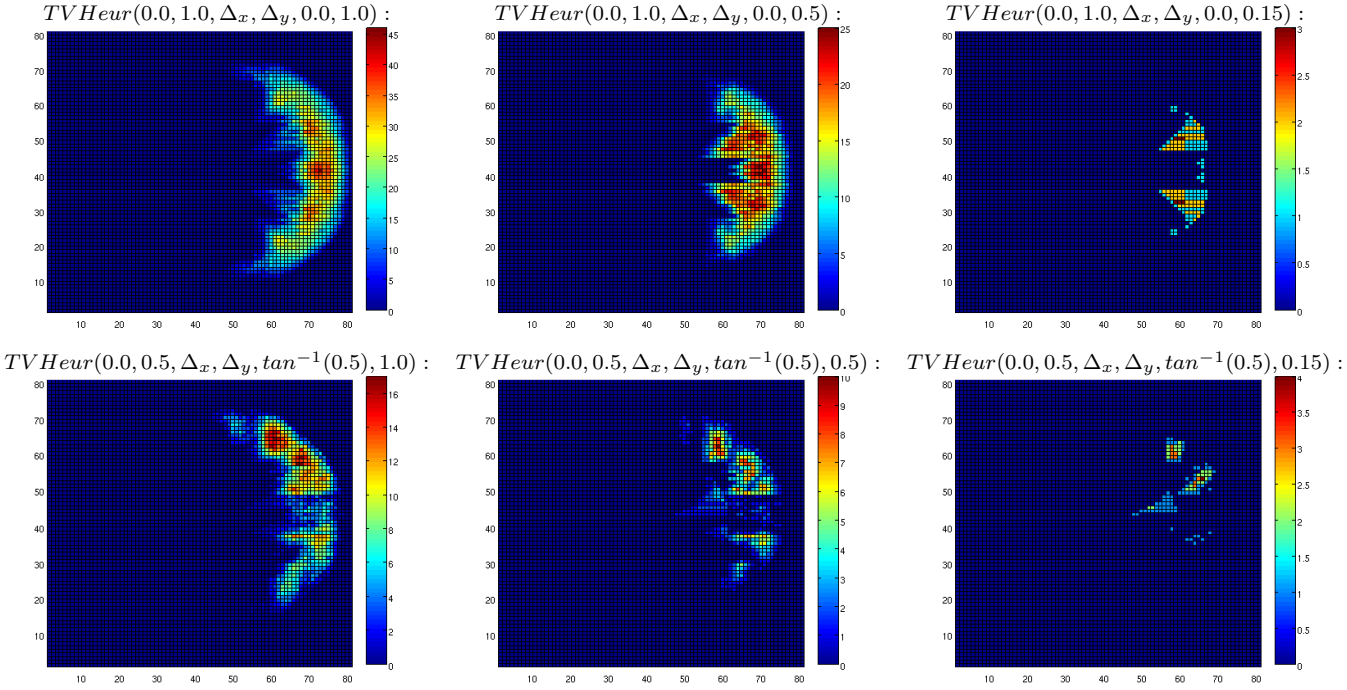

Fig. 2. Time-viable heuristic for a time horizon of $T_{max} = 4.0\,\text{s}$. Displayed are 6 initial-final-state combinations (represented as a subspace of the 6D time-viable heuristic: $TVHeur(a_{start}[rad], v_{start}[m/s], \Delta_x[m/10], \Delta_y[m/10], a_{query}[rad], v_{query}[m/s])$, where 'a' stands for heading, 'v' for velocity, and $\Delta_{x,y}$ for 2D translation between start and query state), with the vehicle origin in the center of the figure facing to the right ($a_{start} = 0.0\,\text{rad}$, $x_{start} = y_{start} = 4.0\,m$). The value of each cell corresponds to the number of time and edge distinct arrivals below T_{max} at that cell's state. The average number of such arrivals increases for larger time horizons.

2) Freespace Heuristic: The freespace heuristic returns the actual constrained cost (i.e time of traversal) of moving between two states in absence of any obstacles, and is generated by performing a Dijkstra search on the state lattice to a finite time horizon. It is thus often highly accurate in the robot vicinity, but becomes less useful further away where obstacles are likely to invalidate it. We precompute it offline and store it in a look-up table. For memory space considerations, the notion of a *trim ratio* was introduced in [19], which specifies the cost ratio between the 2D heuristic and the freespace heuristic. This ratio should be selected as close to 1.0 as memory allows.

3) The Time-Viable Heuristic: The concept of the state lattice postulates that through combination of a (small) finite set of motion primitives, essentially the whole space of feasible motions is described. This implies that the robot can only reach a finite set of states via a limited number of transitions, in bounded time. The time-viable heuristic stores this finite set of states and their predecessor edges in a look-up table. Let us denote the branching factor (a state's average number of successor edges) with BF, and the execution time of the shortest edge (0.1 m) at highest velocity (1.0 m/s) with $t_{shortest}$. The combinatorial number of non-identical trajectories is then computed to be of complexity $BF^{t_{max}/t_{shortest}}$. It thus becomes intractable for large time horizons t_{max}. The limited fidelity of motion predictions bounds this horizon, however, and allows thus for the implementation of such a heuristic in practice. In particular, we precompute offline a 5D Dijkstra search (2D position, heading, velocity, time) on our 4D lattice for every initial state (resulting in queries for heading-velocity combinations only, due to complete invariance in 2D translation, and partial invariance due to rotation). Edges surpassing a fixed time horizon (t_{max}, linked to the fidelity of motion predictions) are discarded. Expansion continues until there are no valid edges left on the queue. As a post-processing step, we perform an optimization to reduce memory usage: due to the lattice's discretization in position (0.1 m) and the robot's maximal velocity (1.0 m/s), a minimal uncertainty in time of 0.1 s results. We exploit this fact and *a posteriori* discretize the time dimension in 0.1 s intervals in order to merge states with identical rounded arrival times and identical predecessor edges together. We then arrive at a reduced size 6D look-up table of time-edge distinct states (see Fig. 2 for an illustration with $t_{max} = 4\,\text{s}$). This heuristic look-up table enables the planning algorithm to determine during search, whether a given expanded state is possibly within range of t_{max} of the search goal state, and if so, with which time instances it needs to be initialized so that it can reach the goal state at $t = 0.0\,\text{s}$.

C. MotionPredictions of Dynamic Obstacles

Predictions concerning the future motion of dynamic obstacles are challenging, as errors in the applied model unfavorably propagate through time. Nonetheless, in the future we would like to add prediction functionality to our model-based pedestrian detection and tracking module [20]. For the remainder of this paper we will assume perfect knowledge of the future motion of dynamic objects (up to t_{max}), however. This allows for a performance analysis in exclusion of artifacts due to motion prediction.

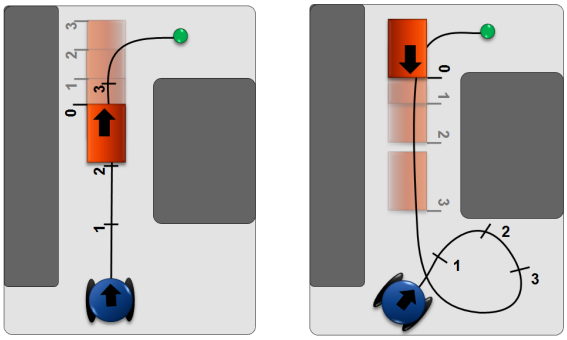

Fig. 3. Schematic of two dynamic scenes (left: scenario II, right: scenario III), where the 2D heuristic and the freespace heuristic are highly inaccurate. The number of state expansions required to find an optimal solution explodes.

D. Selected D* Function Annotations

The D* planning algorithm does not rely on any assumptions concerning the underlying graph but is commonly employed on a 2D grid. In this section we therefore describe adaptations to key functions necessary when planning with motion predictions on a state lattice (see Table II).

1) getTraversalCost(state pred, state succ): In our implementation, the total cost of motion between two states is comprised of a combination of *time of traversal* and *risk of collision* with dynamic obstacles (defined as impact · likelyhood of occupation). Online, risk is stored in a three-dimensional cost map (2D position, time), and updated by combining motion prediction with object labeling.

2) getHeuristicCost(state current): As described in Section II-B, we employ three complimentary heuristics. The heuristic estimate from a state *current* to the goal is then computed as the *maximal* value of the three individual heuristics.

3) getPred(state current): This function returns a list of predecessors of the current state, where predecessors are defined in a graph of directed edges between start and goal location. The introduction of bounded time into the state-space results in two classes of states: these with no time assigned (which signifies that they are not yet within t_{max} of the start configuration), and these associated with a time ε $[0, t_{max}]$. Predecessor expansion is thus divided into three cases:
Case 1: the current state has no time associated with it, and has no predecessors in the range of the time-viable heuristic. In this case, the number of predecessors is equal to the branching factor of the current state. Predecessors are assigned no time.
Case 2: the current state has no time associated with it, but some predecessors are in range of the time-viable heuristic, and thus of t_{max}. This is the most expensive case, as every predecessor in range of the time-viable heuristic is expanded with every stored arrival time at that state. The max. number of arrival times for different configurations is shown in Fig. 2.
Case 3: the current state has a time associated with it. This is the cheapest case, as then *at most* as many edges as the branching factor of the current state are returned.

4) getSucc(state current): This function returns a list of successors of the current state. Successors are assigned an arrival time if and only if the current state also has one assigned and it is lower than t_{max}.

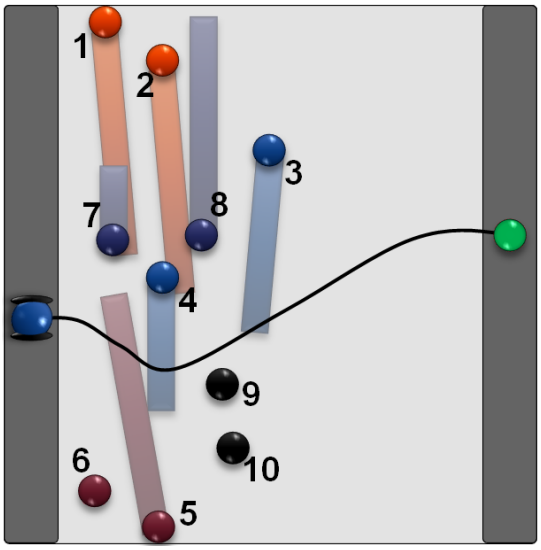

Fig. 4. Schematic of scenario I. An open, but dynamic space of 8x10 m is to be crossed by the robot.

III. Results

A. Complexity Analysis

In Section II-D we described three predecessor expansion cases, where the most expensive one is encountered while crossing the time-viable heuristic's border. In this situation, states with every possible arrival time below t_{max} are put on the priority queue, where they are collision checked against static and dynamic obstacles. These checks (see Table II.A) typically constitute the most costly part of the algorithm. It is thus of great interest to determine the average number of additional states placed on the priority queue relative to a forward searching implementation. Table I summarizes these results for time-viable heuristics of horizon 3.0 s, 3.5 s and 4.0 s, where the search is operated in freespace. It should be noted that in heuristic adverse scenarios, potentially many more crossings (and thus states pushed on the queue) are to be expected (see Section III-B, and Table III below).

B. Simulated Dynamic Scenarios

In this section we compare our high-dimensional D* algorithm to A* in several cluttered and dynamic scenarios. Both algorithms operate on the lattice presented in this paper, and perform cost checks in a dynamic cost field which is based on perfect motion predictions of dynamic objects up to a fixed time horizon (t_{max}). This approach allows us to analyze performance without potentially introducing bias due to inaccurate motion predictions. On the other hand, effects of interaction cannot be analyzed.

1) Scenario I: Scenario I is comprised of an open environment, where the only inaccuracies to the heuristics are due to pedestrians cluttering the scene. The same simulation is run with an increasingly higher number of pedestrians crossing the scene, thus increasing the complexity of the queries: for each query, pedestrians with numbers n <=sceneI.n are taken into account (for numbering, see Table IV and Fig. 4). The

t_{max} $[s]$:	3.0	3.5	4.0
mean	5.2	9.5	16.7
min	1	1	1
max	18	37	62

TABLE I
COMPLEXITY ANALYSIS: AVG. NO. OF ADDITIONAL STATES PLACED ON QUEUE PER EXPANDED NODE UNDER FREE-SPACE ASSUMPTION

robot is required to plan several motions from a random start configuration on the left border to a random goal state on the right border. Table IV shows, that for various prediction horizons and scene complexities, our D* implementation expands a similar number of states to reach the solution as A*. On the other hand, for harder problems, more states are placed on the heap, corresponding to many crossings of the time-viable heuristic's border. These heap placements are costly, as they go along with cost checks and priority queue percolates. In the future, we will attempt to approach this issue through parallelization.

2) Scenarios II & III: Scenarios II and especially III illustrate the need for accurate heuristics when employing high-dimensional heuristic search. Scene II is located in a complex narrow environment, where another (autonomous) vehicle is moving towards the top at a constant velocity of 0.5 m/s (see Fig. 3), and thus blocks the direct path to the goal (green dot). The freespace heuristic is inaccurate due to environmental constraints (walls). The 2D heuristic neglects dynamic obstacles and thus dramatically underestimates the cost to reach the goal. Nonetheless our planner is able to find a solution in fewer than 100 expansion steps.

Scene III is located in the same environment as scene II, except that a dynamic object now moves towards to bottom of the scene at variable speed. The resulting motion requires our vehicle to perform an extra loop to let the obstacle pass, and only then proceed to the goal (see Fig. 3). This extra loop results in many expanded states close to the start position, as the planner believes it is close to the solution, just to discover that at these arrival times, the start state is expected to be blocked by the dynamic object. With our current heuristics, computation of a solution requires > 10000 expansions, and is thus far from real-time. It should be noted that while D* completely fails in such a hard environment, a forward searching algorithm could potentially start executing *partial motions*, combined with frequent re-planning.

getTraversalCost(pred, succ)

```
01 cost = 0;
02 for(tIdx = pred.t; tIdx < succ.t; ++tIdx){
03    (xIdx,yIdx) ← getPosition(pred,succ,tIdx);
04    cost += risk(xIdx,yIdx,tIdx);
05 }
06 cost += timeOfTraversal(pred, succ);
07 return cost;
```

getHeuristicCost(s)

```
01 cost = max(2DHeur(s), freeSpaceHeur(s), timeViableHeur(s));
02 return cost;
```

getPred(s)

```
01 vector<state> allPred;
02 for(predIdx = 0; predIdx<predecessor(s.a,s.v).size(); ++predIdx){
03    pred = predecessor(s.a,s.v,predIdx); {
04    if (s.t == -1 && timeViableHeur(pred).size() != 0){
05       for(tvIdx = 0; tvIdx < timeViableHeur(pred).size(); ++tvIdx)
06          pred.t = timeViableHeur(pred,tvIdx);
07          allPred.push_back(pred);
08       }
09       pred.t = -1;
10       allPred.push_back(pred);
11    } else if (s.t == -1){
12       pred.t = -1;
13       allPred.push_back(pred);
14    } else{
15       pred.t = s.t-pred.tEdge;
16       if (pred.t ε timeViableHeur(pred)){
17          allPred.push_back(pred);
18       }
19    }
20 }
21 return allPred;
```

getSucc(s)

```
01 vector<state> allSucc;
02 for(succIdx = 0; succIdx<successor(s.a, s.v).size(); ++succIdx){
03    succ = successor(s.a,s.v,succIdx);{
04    if (s.t == -1){
05       succ.t = -1;
06       allSucc.push_back(succ);
07    } else{
08       succ.t = s.t + succ.tEdge;
09       allSucc.push_back(succ);
10    }
11 }
12 return allSucc;
```

TABLE II
SELECTED D* FUNCTION ANNOTATIONS

IV. CONCLUSIONS AND FUTURE WORK

In this paper we introduced a novel heuristic, the *time-viable heuristic*. It enables deterministic incremental planning algorithms to be employed in conjunction with motion predictions of dynamic obstacles which are thought to be necessary for planning better solutions in cluttered and dynamic scenes.

We have shown in scenarios of increasing complexity, that for our D* implementation the average number of expanded states is comparable to a forward searching A* query, despite the introduction of the time-viable heuristic. On the other hand, the number of states placed on the priority queue (resulting in priority queue percolates and cost checks, both of them costly operations) grows for our implementation substantially faster with environment complexity.

Future work will thus investigate two directions: first, we will attempt to further increase planning and re-planning speed through parallelization. Second, we will focus on designing more informed, environment aware heuristics in both static and dynamic constrained scenarios.

ACKNOWLEDGMENT

This work has been partially supported by the Swiss National Science Foundation NCCR MC3 and by the European Project EUROPA under contract number FP7-231888.

Scene	I.00		I.02		I.04		I.06		I.08		I.10	
Planner	A^*	D^*	A^*	D^*	A^*	D^*	A^*	D^*	A^*	D^*	A^*	D^*
Motion Prediction Time Bound: 4.0 s												
expansions												
mean	11	17	48	50	61	80	103	97	96	199	107	225
min	9	9	9	9	9	10	15	12	15	18	15	18
max	14	21	290	237	290	361	290	361	300	1684	300	1684
states on heap												
mean	248	145	457	426	601	1044	901	1160	915	3402	989	4782
Motion Prediction Time Bound: 3.5 s												
expansions												
mean	11	11	33	24	44	26	78	39	85	51	95	48
min	9	8	9	8	9	8	15	15	15	15	15	16
max	14	16	270	134	270	134	270	134	300	131	300	131
states on heap												
mean	248	207	416	267	579	284	917	486	976	512	1058	506
Motion Prediction Time Bound: 3.0 s												
expansions												
mean	11	11	30	29	37	29	44	33	50	47	50	47
min	9	8	9	8	9	8	9	8	9	8	9	8
max	14	16	270	261	270	261	270	261	247	233	247	233
states on heap												
mean	248	126	405	227	564	243	593	292	628	378	630	381

TABLE III

SCENARIO I: PERFORMANCE OVERVIEW

REFERENCES

[1] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *Autonomous Mobile Robots: Perception, Mapping, and Navigation (Vol. 1)*, pages 375–382, Los Alamitos, CA, 1991.

[3] M. Likhachev, G. Gordon, and S. Thrun. Ara*: Anytime a* with provable bound son sub-optimality. In *Advances in Neural Information Processing Systems, MIT Press*, 2003.

[4] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. volume 1, pages 968–975 vol.1, 2002.

[5] I. Nesnas M. Pivtoraiko, T. Howard and A. Kelly. Field experiments in rover navigation via model-based trajectory generation and nonholonomic motion planning in state lattices. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS '08)*, February 2008.

[6] R. A. Knepper M. Pivtoraiko and A. Kelly. *Differentially constrained mobile robot motion planning in state lattices*, 26(CMU-RI-TR-):308–333, March 2009.

[7] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2008.

[8] D. Ferguson, T. Howard, and M. Likhachev. Motion Planning in Urban Environments: Part I. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems*, September 2008.

[9] T. Howard D. Ferguson and M. Likhachev. Motion Planning in Urban Environments: Part II. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems*, September 2008.

[10] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '05)*, pages 3231 – 3237, August 2005.

[11] M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in constrained environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[12] R. Rosales and S. Sclaroff. Improved tracking of multiple humans with trajectory prediction and occlusion modeling. In *IEEE CVPR Workshop on the Interpretation of Visual Motion*, 1998.

[13] K. Han and M. Veloso. Physical model based multi-objects tracking and prediction in robosoccer. In *Working notes of the AAAI 1997 Fall Symposium on Model-directed Autonomous Systems*, Boston (US), November 1997.

[14] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *Internationl Journal of Robotics Research*, 24(1):31–48, January 2005.

[15] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1450–1464, September 2006.

[16] L. Liao, D. Fox, and H. Kautz. Learning and inferring transportation routines. In *Proc. of the National Conf. on Artificial Intelligence AAAI-04*, 2004.

[17] A. Kushleyev and M. Likhachev. Time-bounded lattice for efcient planning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[18] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[19] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3375 – 3380, October 2006.

[20] L. Spinello, R. Triebel, and R. Siegwart. Multimodal detection and tracking of pedestrians in urban environments with explicit ground plane extraction. In *Proc. of The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.